CSE 746 - Parallel and High Performance Computing
Lecture 12 - Intel Phi continued

Pawel Pomorski, *HPC Software Analyst*
SHARCNET, University of Waterloo
ppomorsk@sharcnet.ca
http://ppomorsk.sharcnet.ca/

SSH key setup

- need SSH key with forwarding enabled to access Intel Phi cards on SHARCNET
- SSH key will allow you to log in without password
- ssh-keygen -t dsa
- save output to default file if one does not already exist
- pick password (possible to leave blank)
- log in to SHARCNET using your password and add the contents of the public key file (.pub) to ~/.ssh/authorized_keys
- change permissions with chmod 600 ~/.ssh/authorized_keys

SSH key forwarding

- want to be able to go from login node to one of the compute nodes without password
- SSH key forwarding permits this
- ssh -A
- Forwarding needs some setup
- on Mac:
  edit /etc/ssh_config and add

  Host *
  ForwardAgent yes
- on Linux, add this to $HOME/.ssh/config
- run ssh-agent, ssh-add ?

Intel Phi on SHARCNET

- just one node of goblin, node 49 (gb49)
- that node has 2 Intel Phi 5100 series, each 8 GB RAM
- Intel Phi programs can be compiled only on that node
- gb49 does not have standard modules, so before compiling do:
  source /opt/sharcnet/intel/15.0.1/bin/compilervars.sh intel64
  source /opt/sharcnet/intel/15.0.1/mkl/bin/mklvars.sh intel64
  export INTEL_LICENSE_FILE=/opt/sharcnet/intel/15.0.1/license/intel.lic
- need to log into goblin cluster with key forwarding enabled
  ssh -A your_username@goblin.sharcnet.ca
- from goblin, can log into the Intel Phi card with
  ssh gb49-mic0.goblin.sharcnet
  ssh gb49-mic1.goblin.sharcnet

MYO (Virtual-Shared) Memory Model

- alternative to #pragma offload
- similar UVA (Unified Virtual Addressing) in CUDA
- only available in C++, not in Fortran
- MYO means “Mine Yours Ours”
- does not copy all shared variable upon synchronization, only the values that changed between two synchronization points

MYO (Virtual-Shared) Memory Model

- programmer marks variables which need to be shared with _Cilk_shared keyword
- those variables can then be used in host and coprocessor code
- function to be offloaded has to be declared with Cilk_shared keyword, and is offloaded with _Cilk_offload keyword

```
_Cilk_shared int A[N];
…
_Cilk_shared void some_function(){
…  // uses array A
}
…
_Cilk_offload some_function() // uses array A
```

Exercise 1

- write a program to add integer vectors A and B, store result in C
- define A,B,C so that both host and coprocessor can access them
- write function to initialize A and B on host
- write function to do the addition A+B=C on coprocessor
- write function to verify the result on the host
- add syntax to check if the addition is actually running on the coprocessor

Using Multiple Phis

- use OpenMP, have one thread per coprocessor
  see: multicard_multithreaded.c

```
int n_d = _Offload_number_of_devices();
…
#pragma omp parallel for
    for (int i = 0; i < n_d; i++) {
// body of loop is executed by n_d host threads
#pragma offload target(mic:i) inout(response[i:1])
        {
            response[i] = 1;
        }
    }
```

Multiple Phis with one thread

- use OpenMP, have one thread per coprocessor
  see: multicard_onethread.c

```
int n_d = _Offload_number_of_devices();
…

for (int i = 0; i < n_d; i++) {
#pragma offload target(mic:i) inout(response[i:1])
signal(&response[i])
{
      response[i] = 1;
    }
}
for (int i = 0; i < n_d; i++) {
// loop waits for all asynchronous offloads to finish
#pragma offload_wait target(mic:i) wait(&response[i])
}
```

Parallel programming on the Intel Phi

- must have some kind of parallelism to take advantage of the Phi
- there is no automatic parallelization
- serial programs will run slower on the Phi
- the goal is always to do as little programming as possible, and ideally use a library
- the Intel MKL library is a powerful computational tool and it is parallelized for the Intel Phi

Intel MKL

- Math Kernel Library
- highly efficient set of mathematical libraries from Intel
- made to be very easy to use on the Phi
- compiling used to require a complicated set of flags, but with latest version of the compiler 15 it has been simplified
  icc -mkl
  icc -mmic -mkl

Exercise 2a

- look around /opt/sharcnet/mkl and /opt/sharcnet/intel for tutorials, examples, samples
- try for example
  find . -iname "*tutorials*"
- the file you need for this exercise is:
  /opt/sharcnet/mkl/11.1.4/composer_xe_2013_sp1.4.211/Samples/en_US/mkl/tutorials.zip
- find the documentation for the tutorial, download it to your computer
- try the tutorial
- don't use the phi yet, run programs on gb49
- get some timings, experiment with matrix sizes, run "top"

Exercise 2b

- try previous exercises in native mode on the Intel Phi
- compare performance, vary problem size and number of threads
- compile on gb49 with
  icc -mmic -mkl
- need to set:
  export LD_LIBRARY_PATH=/global/c/work/feimao/intel15/mkl/lib/mic/:$LD_LIBRARY_PATH
  these are just libraries copied from:
  /opt/sharcnet/intel/15.0.1/mkl/lib/mic
- run on the MIC with
  OMP_NUM_THREADS=60 ./a.out
- run “top” in another window to observe performance

MKL Automatic Offload

- certain MKL routines can be offloaded to the Intel Phis automatically with no change to the program
- that feature is only enabled for a subset of routines and for problems which are large enough
- need matrix to be big enough for offload to actually happen, see: https://software.intel.com/en-us/articles/intel-mkl-automatic-offload-enabled-functions-for-intel-xeon-phi-coprocessors
- enable offload with:
  export MKL_MIC_ENABLE=1
- monitor offload with:
  OFFLOAD_REPORT=1
  OFFLOAD_REPORT=2

Exercise 2c

- now try automatic offload mode on gb49
- compare performance, vary OMP_NUM_THREADS and MIC_OMP_NUM_THREADS
- need matrix to be big enough for offload to actually happen
- run “top” on the MIC in another window to monitor what is happening

Exercise 2d

- starting with timed code from previous exercise, add offload pragmas and offload , see if speed improves over automatic case